ALGORITMA PARALEL *FP-GROWTH* UNTUK PENGGALIAN KAIDAH ASOSIASI PADA JARINGAN KOMPUTER

F.X. Arunanto, Syaiful Isman
Jurusan Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember
Kampus ITS Sukolilo, Surabaya
Email: anto@if.its.ac.id

ABSTRAK

Algoritma paralel untuk penggalian kaidah asosiasi pada dataset yang besar sangat dimungkinkan dengan tujuan utama untuk mengurangi waktu eksekusi. Semakin besarnya dataset, rata-rata jumlah item dalam sebuah transaksi dan rata-rata jumlah panjang large itemset yang digunakan akan menambah waktu eksekusi dalam penggalian kaidah asosisasi. Oleh karena itu, berbagai algoritma paralel banyak dikembangkan dengan seiringnya waktu, salah satunya adalah algoritma paralel FP-Growth secara trivial paralellization.

Pada penelitian ini, algoritma paralel yang dibentuk akan diimplementasikan terhadap salah satu algoritma penggalian kaidah asosiasi yaitu algoritma FP-Growth. FP-Growth dipilih karena memiliki banyak keuntungan dengan struktur data FP-Tree sebagai bentuk kompresi dataset dan tidak ada waktu yang terbuang untuk perulangan proses pengamatan dataset dibandingkan algoritma sebelumnnya seperti Apriori.

Uji coba dilakukan di lingkungan komputer paralel berbasis jaringan komputer dengan menggunakan Pustaka Massage Passing Interface dan hasilnya membuktikan bahwa algoritma paralel FP-Growth menunjukkan performa yang lebih baik daripada algoritma FP-Growth pada komputer tunggal.

Kata Kunci: *Kaidah Asosiasi, FP-Growth, FP-Tree, Algoritma Paralel.*

1. PENDAHULUAN

Penggalian data merupakan upaya untuk melakukan ekstraksi pola informasi atau pengetahuan yang menarik dari sejumlah besar data. Pola informasi atau pengetahuan dapat dikatakan menarik apabila pola atau pengetahuan tersebut merupakan pola yang non-trivial, implisit, belum diketahui sebelumnya, dan bermanfaat.

Diantara banyak fungsionalitas dari penggalian data, salah satunya adalah penggalian pola asosiasi yang tergolong menarik. Dikatakan menarik karena penggalian pola asosiasi sudah banyak dikenal dan banyak digunakan masyarakat untuk berbagai macam situasi seperti *e-commerce*, klasifikasi, pengelompokan, dan lain sebagainya. Dalam penggalian dan menganalisis pola asosiasi akan ditemukan atribut-atribut yang menunjukkan kondisi dimana atribut-atribut tersebut sering muncul bersamaan dalam suatu data yang diberikan. Penggalian pola asosiasi ini biasanya sering digunakan dalam menganalisis data transaksi. Ada beberapa algoritma yang digunakan dalam upaya untuk memperbaiki kinerja penggalian pola asosiasi tapi algoritma-algoritma tersebut hanya mampu menangani kondisi tertentu saja dan memiliki kekurangan dalam kondisi lainnya. Sedangkan, dalam penggalian data efektivitas dan efisiensi sangatlah diperlukan.

Dalam penelitian ini dilakukan pemodelan algoritma *FP-Growth* pada komputer tunggal menjadi algoritma paralel *FP-Growth* pada komputer lebih dari satu dalam lingkungan paralel. Penelitian yang berkaitan tentang penggalian kaidah asosiasi dan aplikasi paralel akan dijelaskan di bagian berikut ini.

2. PENELITIAN YANG BERKAITAN

Apriori adalah algoritma pertama yang mengawali penggalian kaidah asosiasi pada tahun 1995 menggunakan *frequent pattern* [1]. Sejak saat itu, banyak algoritma baru yang dikembangkan mendasar pada algoritma apriori dalam menggali kaidah asosiasi. Secara *heuristic*, apriori mempunyai performa yang cukup baik dengan mereduksi banyaknya suatu kandidat. Akan tetapi, jika dalam kondisi dengan banyaknya *frequent pattern* atau kecilnya *minimum support threshold* maka untuk algoritma apriori akan menghabiskan waktu yang cukup lama [2]. Karena hal itu, dikembangkan suatu algoritma dengan metode yang dapat menghindari kandidat *generation-and-test* seperti *FP-Growth* .

Sampai saat ini, para peneliti banyak mendesain algoritma paralel pada bidang *datamining* khususnya pada penggalian *frequent pattern* [3-5]. Seperti penggunaan memori secara efisien menggunakan *Hash Partitioned Apriori* (HPA) pada tahun 1996 oleh T. Shintani dan M. Kitsuregawa [6].

3. FP-GROWTH

Algoritma *FP-Growth* dibagi menjadi 2 bagian utama: konstruksi pembuatan stuktur data *FP-Tree* dan pencarian pola-pola item yang *frequent* pada *FP-Tree.*

3.1. Konstruksi Pembuatan*FP-Tree*

Konstruksi pembuatan struktur data *FP-Tree* membutuhkan dua pengamatan pada dataset yang tersimpan informasi transaksi. Pengamatan pertama dilakukan untuk menghitung besarnya *support count* suatu item dan akan di urutkan secara *descending* untuk mendapatkan *F-list*. Pengamatan kedua dilakukan untuk membuat struktur data *FP-Tree*.

Pertama, transaksi pada dataset akan diurutkan berdasarkan *F-list*. Jika terdapat item yang memiliki *support count* lebih kecil dari *minimum support* maka item tersebut akan dipangkas dari transaksi. Selanjutnya akan dibuat node root *FP-Tree* dan transaksi yang sudah terurut dimasukkan ke *FP-Tree*. Pengurutan item-item transaksi berperan penting karena itemset pada *FP-Tree* yang memiliki *prefix* yang sama juga memiliki node yang sama. Jika suatu node sudah terdapat pada transaksi yang telah masuk pada *FP-Tree* atribut *count* pada node tersebut akan bertambah, jika tidak node baru akan dibuat pada *FP-Tree* dengan atribut *count* sama dengan 1.

FP-Tree juga mempunyai *header* tabel yang menyimpan penunjuk-penunjuk suatu node. Penunjuk-penunjuk ini digunakan untuk melewati node pada *FP-Tree* yang digunakan untuk proses penggalian pola-pola item yang *frequent*.

Gambar 1 adalah *algoritma* untuk konstruksi pembentukan *FP-Tree* dengan memanggil fungsi *insert_tree* seperti terlihat pada Gambar 2.

Algoritma 1 (construct *FP-Tree*)

Input: basis data transaksi DB dan min_sup
Output: frequent pattern tree, *FP-Tree*
Method: *FP-Tree*

1. Scan DB satu kali untuk mendapatkan himpunan frequent item F dan support countnya. Urutkan F dalam support descending order L (L merupakan urutan dari frequent item).
2. Membuat node root dari *FP-Tree* T, dan diberi nama *null*.
   Untuk tiap transaksi Trans dalam DB lakukan:
   - Identifikasi dan urutkan frequent items dalam Trans berdasarkan pada L. Frequent items dalam Trans dinotasikan menjadi [p|P], dimana p adalah elemen pertama sedangkan P adalah urutan berikutnya.
   - Panggil fungsi *insert_tree*([p|P], T).

Fungsi *insert_tree*([p|P] , T):

Jika T memiliki child N yang berarti N.item_name = p.item_name, maka nilai count N bertambah 1.

Else buat node baru N, dan count untuk N adalah 1, parent link untuk node N terhubung pada T.

Jika P tidak kosong, maka panggil fungsi *insert_tree*(P, N) secara rekursif.

Gambar 1. Algoritma konstruksi *FP-Tree*

Gambar 2. Fungsi *insert_tree*

3.2. *FP-Growth*

Setelah *FP-Tree* terbentuk sebagai bentuk kompresi informasi transaksi pada dataset, pencarian pola-pola item yang *frequent* menjadi bagian yang mendominasi waktu eksekusi pada algoritma *FP-Growth* .

Pencarian pola-pola item yang *frequent* melibatkan tiga tahapan utama yaitu pembangkitan *conditional pattern base*, pembangkitan *conditional FP-Tree*, dan pencarian *frequent itemset*. Pada hasil akhirnya, *frequent itemset* akan ditemukan dengan urutan *descending* dengan *suffix pattern*.

a) **Tahap Pembangkitan *Conditinal Pattern Base***

Conditional pattern base merupakan subdatabase yang berisikan *prefix path* (himpunan item terurut yang mengawali *k-itemsets*), dan *suffix pattern* (*k-itemsets*). Misalnya, sebuah itemset yang telah terurut berdasarkan *support descending order* {I_6, I_3,I_1, I_{13}, I_{16}}, apabila I_{16} merupakan *suffix pattern*, makaI_6, I_3, I_1, I_{13} adalah *prefix path*nya. Pembangkitan *conditional pattern base* didapatkan melalui *FP-Tree* yang telah dibangun berdasarkan sebuah basis data transaksi.

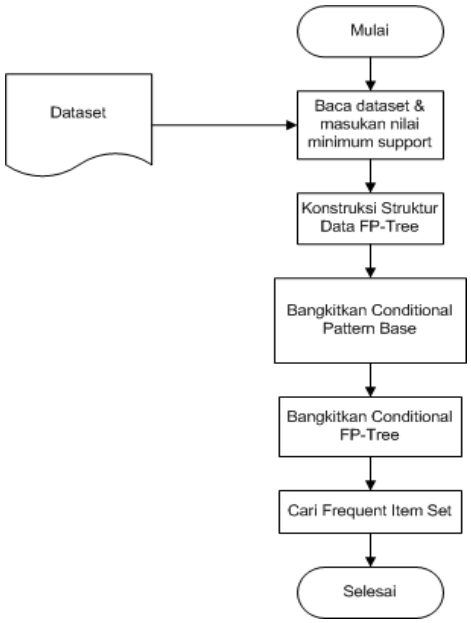
b) **Tahap Pembangkitan *Conditional FP-Tree***

Pada tahap ini, *support count* untuk tiap item pada setiap *conditional pattern base* akan dijumlahkan, kemudian item yang memiliki jumlah *support count* lebih besar sama dengan *min_sup* akan dibangkitkan menjadi sebuah *tree* yang disebut *conditional FP-Tree*.

c) **Tahap Pencarian *Frequent Itemset***

Pada tahap ini, apabila *Conditional FP-Tree* merupakan *single path*, maka akan didapatkan *frequent itemsets* dengan melakukan kombinasi item untuk setiap *Conditional FP-Tree*. Jika bukan single *path* maka, akan dilakukan pembangkitan *FP-Growth* secara rekursif.

Pada Gambar 3 menggambarkan tentang diagram alir algoritma *FP-Growth* pada komputer tunggal.



Gambar 3. Diagram alir algoritma *FP-Growth* pada komputer tunggal

4. ALGORITMA *PARALEL FP-GROWTH* SECARA *TRIVIAL PARALLELIZATION*

Karena proses pada pembangkitan *conditional pattern base* suatu item sebagai bentuk dari proses yang *independent* dengan *conditional pattern base* item yang lain hingga fase pencarian pola-pola item yang *frequent*, maka layak untuk dipertimbangkan dan dikategorikan sebagai proses yang bisa di eksekusi secara paralel.

Pada lingkungan paralel dimana di eksekusinya aplikasi penggalian kaidah asosiasi menggunakan algoritma paralel *FP-Growth* terdapat beberapa asumsi yang perlu diperhatikan, yaitu:

1) Masing-masing komputer memiliki database yang sama.
2) Prosesor Master, yaitu prosesor dengan *ID* prosesor Nol (0) dimana prosesor ini menerima semua hasil pencarian pola-pola item yang *frequent* dari prosesor lainnya.
3) Prosesor Slave, yaitu prosesor dengan *ID* prosesor mulai satu (1) hingga banyaknya prosesor yang digunakan dikurangi satu (seperti identitas array) dimana prosesor ini mengirim semua hasil pencarian pola-pola item yang *frequent* ke prosesor Master.

Secara garis besar berikut ini adalah langkah-langkah algoritma paralel *FP-Growth* :

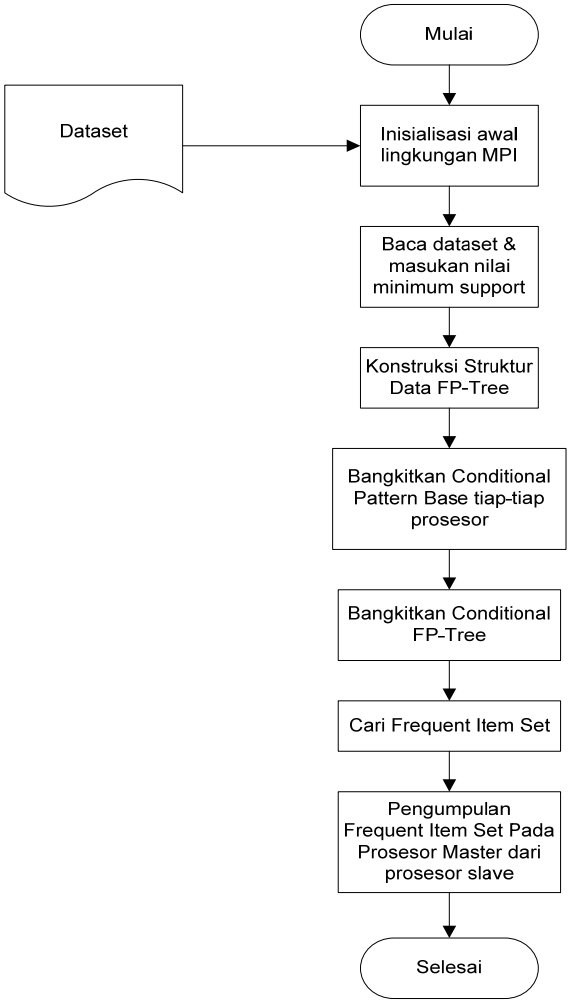
1) Masing-masing komputer melakukan *scan* database dan membangun struktur data *FP-Tree* dan *Flist* sebagai daftar item yang *minimum supportnya* lebih besar sama dengan *minimum support threshold* serta dilakukan pencatatan pointer untuk masing-masing node item ke dalam *head table*.
2) Untuk masing-masing prosesor dengan *ID* prosesor dengan perhitungan (*array number of Flist mod num_proc*) dan hasil sama dengan *ID* prosesor, maka prosesor tersebut melakukan pembangkitan *conditional pattern base* sesuai nomor *array* pada *Flist*.
3) Untuk semua prosesor slave kirim hasil pencarian pola-pola item yang *frequent* ke prosesor master.
4) Untuk prosesor master menerima hasil pencarian pola-pola item yang *frequent* dari prosesor slave berdasar *queue* suatu proses yang masuk terlebih dahulu.
5) Setelah semua hasil pencarian pola-pola item yang *frequent* diterima dari semua prosesor slave, prosesor master akan menggabungkan menjadi satu bentuk solusi yang utuh.

Karena penerimaan hasil pencarian pola-pola item yang *frequent* dilakukan secara *queue*, maka penerimaan dan pengiriman hasil pencarian pola-pola item yang *frequent* sesuai urutan proses pengiriman yang masuk terlebih dahulu ke prosesor master dari semua prosesor slave. Untuk *pseudocode* algoritma paralel *FP-Growth* ditunjukkan pada Gambar 4.

```
Input:database D, minimum support
      min_supp.
Flist = create_flist(D, min_supp);
FPtree = construct_fptree(D, Flist);
If(Flist.size mod num_proc == num_proc)Then
    cond_pbase = build_cond_pbase(FPtree,
    num_Flist);
    cond_fptree =
    construct_cond_fptree(cond_pbase,
    min_supp);
    FP-Growth (cond_fptree, Null)
If(ID prosesor Nol)Then
    While(counter_processor!=(size_processor-
    1))
        receive_frequent_item_set(source_node,
        frequent_item);
Else
    Send_frequent_item_set(dest_node,
    frequent_item_set) ;
```

Gambar 4. Pseudocode algoritma paralel *FP-Growth*

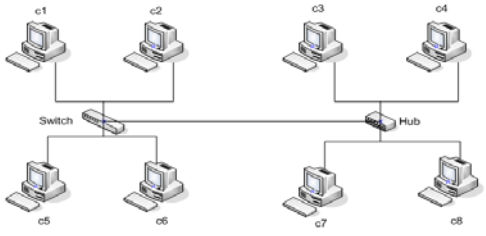
Diagram alir aplikasi algoritma paralel *FP-Growth* secara *trivial parallelization* pada lingkungan paralel MPI akan digambarkan pada Gambar 5.



Gambar 5. Diagram alir algoritma paralel *FP-Growth*

5. HASIL UJI COBA

Lingkungan uji coba menggunakan jaringan komputer (*PC cluster*) yang terdiri dari 8 nodes yangdihubungkan dengan switch fast ethernet 100 MBps dan hub Ethernet 10 MBps seperti terlihat pada Gambar 6.



Gambar 6. Topologi Jaringan Komputer

Setiap node menggunakan processor Intel Pentium IV 3.00 GHz dengan Memori 1 GB dan Sistem Operasi Windows XP dengan aplikasi Microsoft Visual Studio dan DeinoMPI.

Hasil uji coba membandingkan nilai faktor percepatan eksekusi yang didapat dari aplikasi tunggal (*single*) dan paralel untuk seluruh dataset. Untuk menghitung nilai faktor percepatan (*speed-up factor*) menggunakan rumus sebagai berikut:

$$S(n) = \frac{\text{Waktu eksekusi dengan prosesor tunggal}}{\text{Waktu eksekusi dengan n prosesor}} = \frac{t_s}{t_p}$$

dimana t_s dan t_p merupakan waktu eksekusi pada satu dan n prosesor (PC).

Dataset yang dipakai pada uji coba ini menggunakan delapan (8) jenis dataset. Semua dataset ini di dapatkan dari http://illimine.cs.uiuc.edu seperti terlihat pada Tabel 1.

Tabel 1. Spesifikasi Dataset

Nama	Rata-rata item per transaksi	Rata-rata large item set	Jumlah Transaksi
T20I20D10K	20	20	7447
T20I20D100K	20	20	74893
T20I25D10K	20	25	5919
T20I25D100K	20	25	60484
T25I20D10K	25	20	8867
T25I20D100K	25	20	89037
T25I25D10K	25	25	7395
T25I25D100K	25	25	74574

Dalam dataset ini, ukuran rata-rata itemset per transaksi dan ukuran rata-rata *large itemset* di-set 20 dan 25. Sedangkan jumlah transaksi dalam dataset di-set 100K dan 10K.

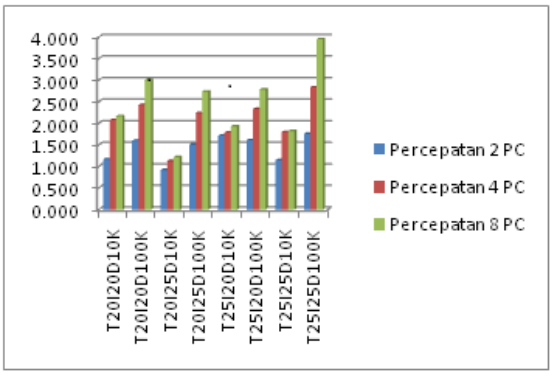
Uji coba dilakukan untuk melihat perubahan waktu yang dibutuhkan untuk mendapatkan *frequent itemset* dengan minimum support sebesar 0.8%. Hasil eksekusi algoritma tunggal dan paralel *fp-growth* untuk setiap dataset dapat dilihat pada Tabel 2. Sedangkan Tabel 3 menunjukkan faktor percepatannya.

Tabel 2. Waktu Eksekusi

Dataset	Jumlah Komputer			
	1 pc	2 pc	4 pc	8 pc
T20I20D10K	28.302	24.161	13.634	13.036
T20I20D100K	28.442	17.763	11.716	9.463
T20I25D10K	3.828	4.161	3.397	3.142
T20I25D100K	21.190	14.066	9.448	7.737
T25I20D10K	40.548	23.641	22.625	21.023
T25I20D100K	81.808	50.724	35.030	29.363
T25I25D10K	586.177	511.980	325.413	321.930
T25I25D100K	52.204	29.688	18.416	13.217

Tabel 3. Nilai faktor percepatan algoritma paralel *FP-Growth*

dataset	percepatan		
	2 pc	4 pc	8 pc
T20I20D10K	1.171	2.076	2.171
T20I20D100K	1.601	2.428	3.006
T20I25D10K	0.920	1.127	1.218
T20I25D100K	1.506	2.243	2.739
T25I20D10K	1.715	1.792	1.929
T25I20D100K	1.613	2.335	2.786
T25I25D10K	1.145	1.801	1.821
T25I25D100K	1.758	2.835	3.950



Gambar 7. Grafik percepatan algoritma paralel *FP-Growth*

Dari hasil uji coba, lihat Tabel 3, untuk eksekusi paralel dengan 2 prosesor pada dataset T20I25D10K membutuhkan waktu lebih lama dari waktu eksekusi sekuensial dengan menunjukkan nilai faktor percepatan kurang dari 1. Hal ini disebabkan karena waktu eksekusi didominasi oleh waktu komunikasi antar prosesor. Sedangkan untuk dataset yang lain faktor percepatannya sebanding dengan jumlah prosesor.

Pada Gambar 7 menunjukkan peningkatan faktor percepatan untuk jumlah prosesor yang lebih besar, hal ini sejalan dengan tujuan penggunaan algoritma paralel.

6. KESIMPULAN

Algoritma paralel *fp-growth* untuk penggalian kaidah asosiasi telah diimplementasikan pada jaringan komputer dengan jumlah PC 8 buah, jika jumlah komputer (prosesor) yang digunakan lebih banyak lagi maka peningkatan faktor percepatannya akan terlihat lebih signifikan.

Untuk jumlah transaksi yang besar, penggunaan algoritma paralel *fp-growth* untuk penggalian kaidah asosiasi lebih baik dibandingkan dengan algoritma tunggal. Akan tetapi untuk jumlah transaksi kecil peningkatan faktor percepatan kurang signifikan bahkan ada yang kurang dari 1.

7. DAFTAR PUSTAKA

[1] R. Agrawal dan R. Srikant. *"Fast Algorithms for Mining Association Rules"*. Pada Proceedings ke-20 International Conference di VLDB, pp. 487-499, September 1994.

[2] R.Agrawal, C. Aggarwal, dan V. V. V. Prasad. *"A tree projection algorithm for generation of frequent itemsets"*. Pada J. Parallel dan Distributed Computing, 2000.

[3] Han. J., Pie, J., Yin, Y., Mining Frequent Pattern without Candidate Generation, *School of Computing Science, Simon Fraser University*, 2000.

[4] R. Agrawal dan J. C. Shafer. *"Parallel Mining of Association Rules"*. Pada IEEE Transaction on Knowledge and Data Engineering, Vol. 8, No. 6, pp. 962-969, December, 1996.

[5] J.S.Park, M.-S.Chen, P.S. Yu *"Efficient Parallel Algorithms for Mining Association Rules"* Pada Proc. ke-4 International Conference on Information and Knowledge Management (CIKM'95), pp.31-36, November, 1995.

[6] T. Shintani dan M. Kitsuregawa *"Hash Based Parallel Algorithms for Mining Association Rules"*. Pada IEEE Fourth International Conference on Parallel and Distributed Information Systems, pp. 19-30, December, 1996.

[7] Iko Pramudiono dan M. Kitsuregawa "Parallel FP-Growth on PC cluster", 2003.